Extended Backus–Naur Form

From Wikipedia, the free encyclopedia

In computer science, **Extended Backus–Naur Form (EBNF)** is a metasyntax notation used to express context-free grammars: that is, a formal way to describe computer programming languages and other formal languages. It is an extension of the basic Backus–Naur Form (BNF) metasyntax notation.

EBNF was originally developed by Niklaus Wirth. However, many variants of EBNF are in use. The International Organization for Standardization has adopted an EBNF standard (ISO-14977). This article describes EBNF as specified by the ISO. Other EBNF variants use somewhat different syntactic conventions.

Contents

Basics

A code, e.g. source code of a computer program consisting of terminal symbols—that is, visible characters, digits, punctuation marks, white space characters etc.

The EBNF defines production rules where sequences of symbols are respectively assigned to a nonterminal:

```
digit excluding zero = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
digit                = "0" | digit excluding zero ;
```

This production rule defines the nonterminal *digit* which is on the left side of the assignment. The vertical bar represents an alternative and the terminal symbols are enclosed with quotation marks followed by a semicolon as terminating character. Hence a *Digit* is a *0* or a *digit excluding zero* that can be *1* or *2* or *3* and so forth until *9*.

A production rule can also include a sequence of terminals or nonterminals, each separated by a comma:

```
twelve                          = "1" , "2" ;
two hundred one                 = "2" , "0" , "1" ;
three hundred twelve            = "3" , twelve ;
twelve thousand two hundred one = twelve , two hundred one ;
```

Expressions that may be omitted or repeated can be represented through curly braces { ... }:

```
natural number = digit excluding zero , { digit } ;
```

In this case, the strings *1*, *2*, ...,*10*,...,*12345*,... are correct expressions. To represent this, everything that is set within the curly braces may be repeated arbitrarily often, including not at all.

An option can be represented through squared brackets [...]. That is, everything that is set within the square braces may be present just once, or not at all:

```
integer = "0" | [ "-" ] , natural number ;
```

Therefore an integer is a zero (*0*) or a natural number that may be preceded by an optional minus sign.

EBNF also provides, among other things the syntax to describe repetitions of a specified number of times, to exclude some part of a production, or to insert comments in an EBNF grammar.

Motivation to extend the BNF

The BNF had the problem that options and repetitions could not be directly expressed. Instead, they needed the use of an intermediate rule or alternative production defined to be either nothing or the optional production for option, or either the repeated production or itself, recursively, for repetition. The same constructs can still be used in EBNF.

Option:

```
signed number = [ sign ] , number ;
```

can be defined in BNF style as:

```
<signed number> ::= <sign> , <number> | <number> ;
```

or

```
<signed number> ::= <optional sign> , <number> ;
<optional sign> ::= ε | <sign> ; (* epsilon ("ε") is used to denote more clearly an empty production *)
```

Repetition:

```
number = { digit } ;
```

can be defined in BNF style as:

```
<number> ::= ε | <number> , <digit> ;
```

Other additions and modifications

The EBNF eliminates some of the BNF's flaws:

- The BNF uses the symbols (<, >, |, ::=) for itself. When these appear in the language that is to be defined, the BNF cannot be used without modifications and explanation.
- A BNF syntax can only represent a rule in one line.

The EBNF solves these problems:

- Terminals are strictly enclosed within quotation marks ("..." or '...'). The angle brackets ("<...>") for nonterminals can be omitted.
- A terminating character, the semicolon, marks the end of a rule.

Furthermore there are mechanisms for enhancements, defining the number of repetitions, excluding alternatives (e.g. all characters excluding quotation marks), comments, etc.

Despite all enhancements, the EBNF is not "more powerful" than the BNF in the sense of the language it can define. As a matter of principle any grammar defined in EBNF can also be represented in BNF. However, this often leads to a considerably larger representation.

The EBNF has been standardized by the ISO under the code *ISO/IEC 14977:1996(E)*.

Under some circumstances, any extended BNF is referred to as EBNF. For example, the W3C uses *one* EBNF to specify XML.

Another example

A simple programming language that allows only assignments can be defined in EBNF as follows:

```
(* a simple program in EBNF - Wikipedia *)
program = 'PROGRAM' , white space , identifier , white space ,
          'BEGIN' , white space ,
          { assignment , ";" , white space } ,
          'END.' ;
identifier = alphabetic character , { alphabetic character | digit } ;
number = [ "-" ] , digit , { digit } ;
string = '"' , { all characters - '"' } , '"' ;
assignment = identifier , ":=" , ( number | identifier | string ) ;
alphabetic character = "A" | "B" | "C" | "D" | "E" | "F" | "G"
                     | "H" | "I" | "J" | "K" | "L" | "M" | "N"
                     | "O" | "P" | "Q" | "R" | "S" | "T" | "U"
                     | "V" | "W" | "X" | "Y" | "Z" ;
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
white space = ? white space characters ? ;
all characters = ? all visible characters ? ;
```

A syntactically correct program then would be:

```
PROGRAM DEMO1
BEGIN
  A0:=3;
  B:=45;
  H:=-100023;
  C:=A;
  D123:=B34A;
  BABOON:=GIRAFFE;
  TEXT:="Hello world!";
END.
```

The language can easily be extended with control flows, arithmetical expressions and Input/Output instructions. Then a small, usable programming language would be developed.

The following characters that are proposed in the standard as normal representation have been used:

Usage	Notation
definition	**=**
concatenation	**,**
termination	**;**
separation	**|**
option	**[...]**
repetition	**{ ... }**
grouping	**(...)**
double quotation marks	**" ... "**
single quotation marks	**' ... '**
comment	**(* ... *)**

special sequence	? ... ?
exception	-

Conventions

1. The following conventions are used:

- Each meta-identifier of Extended BNF is written as one or more words joined together by hyphens;
- A meta-identifier ending with "-symbol" is the name of a terminal symbol of Extended BNF.

2. The normal character representing each operator of Extended BNF and its implied precedence is (highest precedence at the top):

```
* repetition-symbol
- except-symbol
, concatenate-symbol
| definition-separator-symbol
= defining-symbol
; terminator-symbol
```

3. The normal precedence is overridden by the following bracket pairs:

```
'  first-quote-symbol            first-quote-symbol  '
"  second-quote-symbol          second-quote-symbol  "
(* start-comment-symbol          end-comment-symbol *)
(  start-group-symbol              end-group-symbol  )
[  start-option-symbol            end-option-symbol  ]
{  start-repeat-symbol            end-repeat-symbol  }
?  special-sequence-symbol   special-sequence-symbol ?
```

The first-quote-symbol is the apostrophe as defined by ISO/IEC 646:1991, that is to say Unicode 0x0027 ('); however, the font used in ISO/IEC 14977:1996(E) renders it very much like the acute, Unicode 0x00B4 (´), so confusion sometimes arises.

As examples, the following syntax rules illustrate the facilities for expressing repetition:

```
aa = "A";
bb = 3 * aa, "B";
cc = 3 * [aa], "C";
dd = {aa}, "D";
ee = aa, {aa}, "E";
ff = 3 * aa, 3 * [aa], "F";
gg = {3 * aa}, "G";
```

Terminal strings defined by these rules are as follows:

```
aa: A
bb: AAAB
cc: C AC AAC AAAC
dd: D AD AAD AAAD AAAAD etc.
ee: AE AAE AAAE AAAAE AAAAAE etc.
ff: AAAF AAAAF AAAAAF AAAAAAF
gg: G AAAG AAAAAAG etc.
```

EBNF Extensibility

According to the ISO 14977 standard EBNF is meant to be extensible, and two facilities are mentioned. The first is part of EBNF grammar, the special sequence, which is arbitrary text inside question marks, whose interpretation is beyond the scope of the EBNF standard. For example, the space character could be defined by

the following rule:

```
space = ? US-ASCII character 32 ?;
```

The second is using the fact that parentheses cannot in EBNF be placed next to an identifier. The following is not valid EBNF:

```
something = foo ( bar );
```

So an extension of EBNF could use that notation. For example, in a Lisp grammar, function application could be defined by the following rule:

```
function application = list( symbol , [ { expression } ] );
```

Related work

- The W3C used a different EBNF to specify the XML syntax.
- The British Standards Institute published a standard for an EBNF: BS 6154 in 1981.
- The IETF uses Augmented BNF (ABNF), specified in RFC 5234.

See also

- Augmented Backus–Naur Form
- Backus–Naur Form
- Regular expression
- Spirit Parser Framework
- Wirth syntax notation
- Phrase structure rules, the direct equivalent of EBNF in natural languages.

References

- Niklaus Wirth: What can we do about the unnecessary diversity of notation for syntactic definitions? CACM, Vol. 20, Issue 11, November 1977, pp. 822–823.
- Roger S. Scowen: Extended BNF — A generic base standard. Software Engineering Standards Symposium 1993.
- The International standard (ISO 14977) that defines the EBNF is now freely available as Zip-compressed PDF file.

External links

- Article "EBNF: A Notation to Describe Syntax (PDF)" by Richard E. Pattis describing the functions and syntax of EBNF
- Article "BNF and EBNF: What are they and how do they work?" by Lars Marius Garshol
- Article "The Naming of Parts" by John E. Simpson
- ISO/IEC 14977 : 1996(E)
- RFC 4234 – Augmented BNF for Syntax Specifications: ABNF
- BNF/EBNF variants – a table by Pete Jinks comparing several syntaxes.
- Create syntax diagrams from EBNF
- EBNF Parser & Renderer

This article was originally based on material from the Free On-line Dictionary of Computing, which is licensed under the GFDL.

Categories: Formal languages | Compiler theory

Hidden categories: Wikipedia articles incorporating text from FOLDOC